2025 IEEE 26th International Conference of Young Professionals in Electron Devices and Materials (EDM) | 978-1-6654-7737-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/EDM65517.2025.11096772

ModuSLAM: a Modular Framework for Factor Graph-based Localization and Mapping

Mark Griguletskii
Department of digital engineering
Skolkovo institute of science and technology
Moscow, Russia
mark.griguletskii@skoltech.ru

Pavel Osinenko
Department of digital engineering
Skolkovo institute of science and technology
Moscow, Russia
p.osinenko@skoltech.ru

***Abstract*—Modern autonomous vehicles and robots rely on versatile sensors for diverse localization and mapping tasks. Factor graphs serve as a powerful approach for consistent sensor fusion, enabling maximum a posteriori state estimation. Recent advancements in LiDAR and Radar point cloud mapping, visual-inertial odometry, bundle adjustment, and rendering require the integration of complex methods into a unified pipeline. Such integration necessitates a software architecture that is modular, scalable, and easily modifiable. This paper addresses key software design challenges in localization and mapping frameworks. It introduces a hybrid architecture combining hierarchical tree-based and flat service-based structures to streamline development, testing, and application. The introduced data structures facilitate efficient module interaction, enabling seamless integration of new functionalities. The resulting design ensures flexibility and adaptivity, supporting various algorithms across different tasks while catering to a broad research community. The framework is designed to accommodate emerging trends, such as semantic mapping and hierarchical environment representations. Our open-source library, implemented in Python, is publicly available at https://github.com/fatrybl/ModuSLAM.**

***Keywords*—framework, SLAM, factor graph, mapping, localization, smoothing, robotics, navigation.**

I. Introduction

With the growing popularity of autonomous systems, the demand for accurate and reliable localization and mapping algorithms is increasing. Simultaneous Localization and Mapping (SLAM) is a well-known problem in robotics and computer vision, aiming to estimate robot's current state parameters (localization) and the properties of the observed environment (mapping). Given noisy sensor measurements Z, this problem can be formulated as a maximum a posteriori (MAP) inference, where the goal is to maximize the posterior density $p(X|Z)$ of the state variables X that best explain the observed measurements Z:

$$X^{\text{MAP}} = \arg\max_X p(X|Z) = \arg\max_X \frac{p(Z|X)\,p(X)}{p(Z)} \quad (1)$$

One of the most common approaches is to use a factor graph-based representation [1] of an inference model, where state nodes are connected by constraints (edges) via factor nodes. This approach is widely adopted in robotics due to its flexibility and scalability. However, implementing factor graph-based SLAM algorithms from scratch is challenging, as it requires a deep understanding of optimization theory and extensive coding. Many researchers prefer to focus on developing new solutions and testing hypotheses on real-world datasets without significant modifications to existing codebases. While most open-source frameworks [2] achieve qualitative results, they are often tailored to specific sensor configurations and applications, such as camera tracking, point cloud mapping or bundle adjustment. This specialization, combined with rigid architecture, makes integrating new approaches or modifying existing solutions difficult. Our work addresses the problem of software design patterns for localization & mapping frameworks to ensure applicability across diverse tasks and presents ModuSLAM — an open-source, modular Python framework for various SLAM applications. The pipeline and communication between main modules are illustrated in Fig. 1. The architecture combines hierarchical and flat service-based design patterns, enabling easy extension, modification, and testing of new modules. The framework is not limited to classical geometrically constrained problems but can also adapt to emerging trends, including semantic and hierarchical environment representations.

II. Related Work

TABLE I: Comparison of Open-source SLAM Frameworks Based on Various Criteria: Maintenance Status (Update), Python Support, Multi-sensor Capabilities, Scalability (T: Task-driven, G: General-purpose), and Backend Optimization.

Method	Update	Python supp.	Multi-Sensor	Scale	Backend
pySLAM	✓	✓	–	T	pyg2o*
Plug-and-Play	–	–	✓	G	custom
WOLF	✓	–	✓	G	ceres
maplab 2.0	–	–	✓	G	ceres
MOLA	✓	–	–	T	gtsam
RTAB-Map	✓	–	✓	G	ceres/toro/g2o/gtsam
ModuSLAM	✓	✓	✓	G	gtsam

Table I provides a comparison of various aspects of open-source SLAM frameworks. The "Update" status indicates whether a source code has been updated within the past year. The "Python supp." column highlights the availability of a Python API, which is particularly relevant for researchers who prefer implementing solutions in Python. The "Multi-Sensor" column specifies the library's capability to integrate multiple

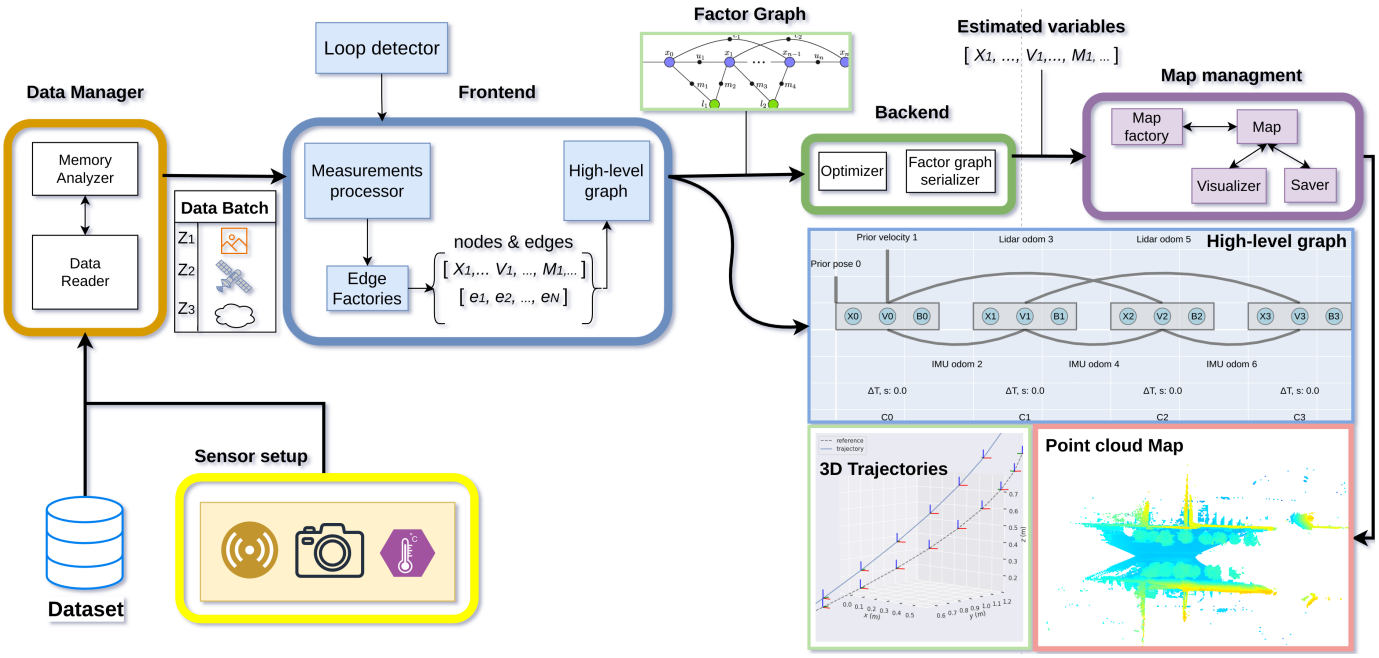


Fig. 1: Block diagram of the main modules' functionality and their communication in ModuSLAM for the mapping process.

sensors without requiring significant modifications to the existing codebase or interfaces. The "Scale" column distinguishes between task-driven frameworks, designed for specific applications with fixed sensor configuration, and general-purpose frameworks suitable for a wide range of SLAM problems. The "Backend" column specifies the optimization library used, as different backends implement state-of-the-art optimization algorithms with varying features and performance levels. The * symbol next to "pyg2o" indicates that it has not been updated for a long time, while the g2o [3] is actively maintained. Since this paper focuses on general approaches applicable to diverse SLAM applications, the frameworks most closely aligned with ModuSLAM in these aspects are WOLF and RTAB-Map.

For Visual SLAM challenges, pySLAM [4] provides a comprehensive suite of processing tools and features for state estimation using monocular, stereo, and RGB-D cameras. It offers a flexible interface for integrating both classical and modern neural net-based visual features, along with support for loop closure methods, volumetric reconstruction, and depth prediction. Additionally, it includes integration with Gaussian splatting for 3D rendering and visualization. However, as it is primarily focused on visual SLAM, it lacks support for other sensors such as IMU, GPS, LiDAR, and Radar. The optimization in backend relies on the "g2opy" library, which has not been maintained for the past eight years and lacks many modern factor graph optimization features. PySLAM uses Python 3.8, missing out on the latest features, improvements for fast and efficient computation, and stricter type checking.

The authors of Plug-and-Play SLAM [5] library suggest to decompose modules into core and support based on their tasks in tree-based fashion. Cores modules are responsible for raw data preprocessing, relative motion estimation, and factor graph creation. Support modules perform loop detections, correspondences association and backend optimization. The paradigm of task-driven modules decompositions is similar to our approach. However, the authors are more focused on pose-graph problem for poses estimation. The provided schemes of Multi-Aligner and Multi-Tracker modules contain sequential steps that can not be omitted for non pose-based applications. Meanwhile the roles of modules are well-described, the interaction between them is not clear enough. Since the library has not been maintained for last 5 years, this challenges the possibility of using it with state-of-the-art algorithms.

The authors of WOLF [6] propose a tree-based modular architecture for SLAM frameworks, centered around the WOLF-tree: a modular structure designed for easy extension and modification. They demonstrate its versatility in visual-inertial odometry, online calibration, and LiDAR mapping with GPS. A key feature is the clear representation of module interactions as a graph, which can be serialized into a factor graph for state estimation. However, pure tree-based architectures have limitations: deep, unbalanced trees can hinder module communication; a single point of failure can disrupt the entire tree; and the hierarchical structure may lack adaptability for changes requiring a more flexible or flat design. In our work, we introduce a hybrid architecture combining hierarchical tree-based structures—essential for logic control—with a flat service-based design. This approach allows modules to be easily configured, modified, reused, and launched as independent processes.

The second version of Maplab [7], designed for multi-agent SLAM, improves modularity and scalability. It supports various sensors and algorithms for visual and LiDAR-based navigation, integrating external odometry approaches like ROVIO [8], OKVIS [9], and FAST-LIO-2 [10]. However,

reusing the same sensor data, such as LiDAR point clouds for both FAST-LIO-2 odometry and ICP-based local map refinement, introduces correlated measurements. This conflicts with the requirement of measurement independence, essential for consistent inference through multiplying density functions of uncorrelated variables. Equation 2 in Section III highlights the need for this property. While the framework excels at aggregating standalone algorithms and providing communication interfaces, it lacks an architecture for developing new approaches. Although it allows to achieve qualitative results faster, it might pose challenges in obtaining consistent estimations for complex tasks involving multiple aggregated algorithms.

The MOLA [11] framework focuses on LiDAR odometry, localization, and map manipulation, advocating for pure point cloud-based SLAM solutions with qualitative long-range mapping results. It employs loop closures and GPS for globally consistent estimations but limits the use of additional sensors like IMUs, wheel encoders, and cameras. In contrast, RTAB-Map [12] supports multi-sensor fusion, including LiDARs, and integrates various graph solvers such as GTSAM [13], g2o [3], ceres [14], and TORO [15]. It can also leverage sensors from iOS and Android devices for localization and mapping. Despite 11 years of development, RTAB-Map's documentation primarily showcases application-specific combinations of approaches, offering limited insight into its architecture and module interactions. This poses challenges for researchers adapting the library to specific tasks. Originally designed as a visual SLAM algorithm, RTAB-Map retains legacy design artifacts, hindering flexibility and adaptation to emerging technologies. Its functionality is concentrated in two large classes, complicating code reuse. Furthermore, the pose-graph generation interface supports only odometry and visual/LiDAR data, lacking external loop closure or proximity constraints, which limits adaptability and integration of alternative sensors like GPS or WiFi-based systems.

The majority of existing libraries do not prioritize the diversity of state estimation problems. Beyond poses, velocities, and IMU biases, factor graphs can estimate various parameters, such as external/internal sensor calibrations, object semantic properties, the parameters of 3D Gaussian distributions for rendering [16], or time offset [17] between measurements. Additionally. the new era of SLAM integrates classical geometric constraints with semantic ones, demanding frameworks that are adaptable to these evolving requirements. These two challenges should be taken into account while designing the software design for modular and application-agnostic framework. In the following section, we describe the architectural enhancements implemented in ModuSLAM, which aim to improve the flexibility and scalability of the SLAM framework.

III. Methodology

A. Architecture

In many practical applications the pipeline for state estimation typically remains unvarying: **1.** collect and process measurements, **2.** create and add factors to the factor graph, and **3.** solve the optimization problem to obtain state estimations.

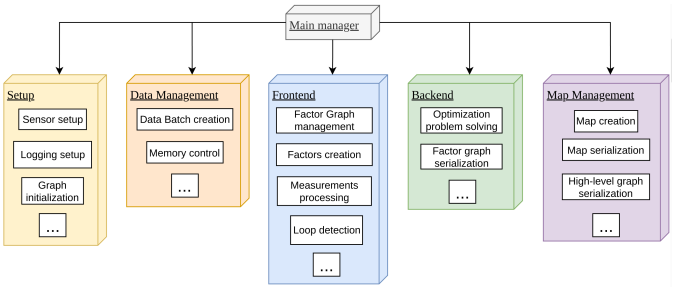


Fig. 2: Functional scheme of the main modules in ModuSLAM, highlighting their primary roles.

Fig. 1 illustrates the complete workflow, from data collection to the generation of the final point cloud map. In our work, we propose a combined architecture for the SLAM framework that integrates a hierarchical (tree-based) upper level with a flat service-based lower level. The upper level manages process control and module communication, encompassing the core logic for solving the SLAM problem. The lower level consists of independent modules, which are implemented as services to execute user-defined logic. Each module can be configured and launched as a separate process. The configuration involves two steps: YAML files parsing and data types validation.

Fig. 2 illustrates the five main blocks and their roles, overseen by the Main Manager. Three dots in each block indicate extensibility with additional functionality. Users can define a custom Main block with task-specific logic and instances. The Setup block initializes application-specific modules, including logging configuration, sensor initialization, and other preparatory tasks. The graph initialization procedure establishes prior factors for the factor graph. The Data Management block retrieves raw measurements from datasets, stores them in the `Data Batch`, and monitors computational resources to prevent memory overflow and crashes. The Frontend handles graph manipulations, such as factor expansion, truncation, and node marginalization, while distributing raw measurements to processing modules and aggregating their outputs. The Backend performs optimization using various solvers and serializes the factor graph for debugging. The Map Management block generates, saves, and renders maps from estimated variables. Each module can run multiple parallel instances with distinct configurations. While most classes are instance-oriented, some serve as shared state (singleton-like) storage accessible to other modules.

The proposed design pattern effectively addresses most SLAM challenges. For point cloud-based, visual-inertial, or object-based mapping, the main logic control principles are very similar, and can be managed by the Main block. However, during the localization & mapping process the modules might require additional information. The appearance-based loop closure detector requires camera images and/or LiDAR/Radar scans to verify visited locations. To gather this data, the module must instantiate a component responsible for collecting measurements, which we refer to as the `Batch Factory`. Multiple instances of this module can be utilized asynchronously without conflicts. Similarly, the Mapping

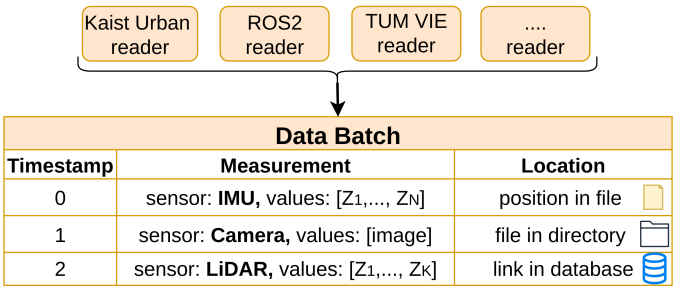


Fig. 3: `Data Batch` is a standardized structure for storing raw measurements collected by dataset-specific Data Readers.

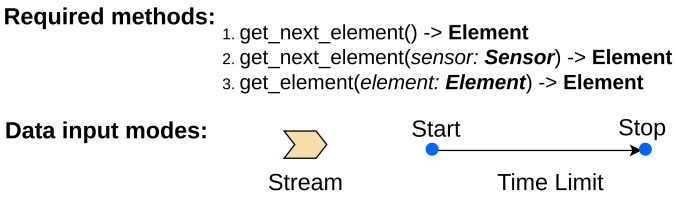


Fig. 4: Essential properties for a custom Data Reader to ensure compatibility with ModuSLAM.

block requires not only estimated variables but raw scans or images at specific timestamps to construct the map. For this purpose, a separate instance of the `Batch Factory` with its own configuration is needed. This instance-based design pattern enables the creation of uniquely configured objects for specific tasks, facilitating efficient utilization of other modules.

B. Data Management

As illustrated in Fig. 3, we propose storing raw measurements in a `Data Batch`. This structure combines a double-ended queue and a set, enabling efficient addition and removal of items at the front or end, as well as existence checks with $O(1)$ complexity. Each element in a `Data Batch` includes a timestamp, a measurement, and a location descriptor. The timestamp ensures proper data ordering. The measurement consists of raw data (such as a point cloud(s), image(s), 3D object(s), or any other type) and the sensor that captured it. The location descriptor specifies the measurement's source within the dataset, allowing a loop detector or any other module to request the actual measurement using this reference. Raw measurements are retained only during the processing stage and are removed once the corresponding factors are created.

In ModuSLAM, we emphasize the diversity of datasets used to test various algorithms. To support this, a custom Data Reader for each dataset must adhere to a standardized interface. As shown in Fig. 4, every Data Reader module should provide measurements in two modes: Stream and Time Limit. The Stream regime delivers measurements sequentially from the dataset without time constraints until the dataset is exhausted, while the Time Limit mode supplies measurements within a specific time range, with start and stop timestamps defined in the configuration file. Since read-write operations are typically time-consuming, it is more efficient to collect measurements in batches. However, certain SLAM applications require the Data Reader to iteratively provide single measurement without any aggregations. To ensure consistent functionality across datasets, the Data Reader must implement three core methods:

1) `get_next_element` — iteratively reads the next measurement from a dataset. The next measurement is defined as the one with the earliest timestamp in ascending order.
2) `get_next_element(sensor: Sensor)` — iteratively reads the next measurement from a dataset for the specified sensor.
3) `get_element(element: Element)` — retrieves the raw measurement using the `Location` attribute of the given input element that lacks raw data.

All three methods return an object of type `Element` with the required properties and raw measurement. This functionality is essential for other modules, such as the loop detector or mapping module, which require raw sensor data.

C. High-level Graph

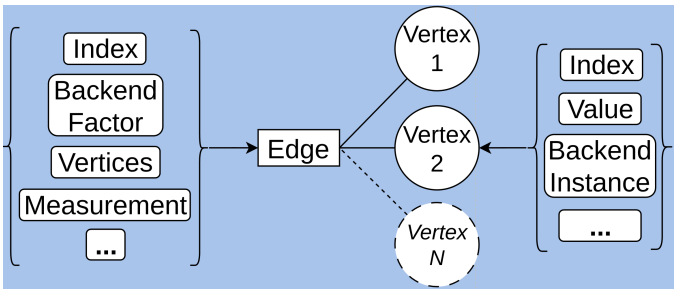
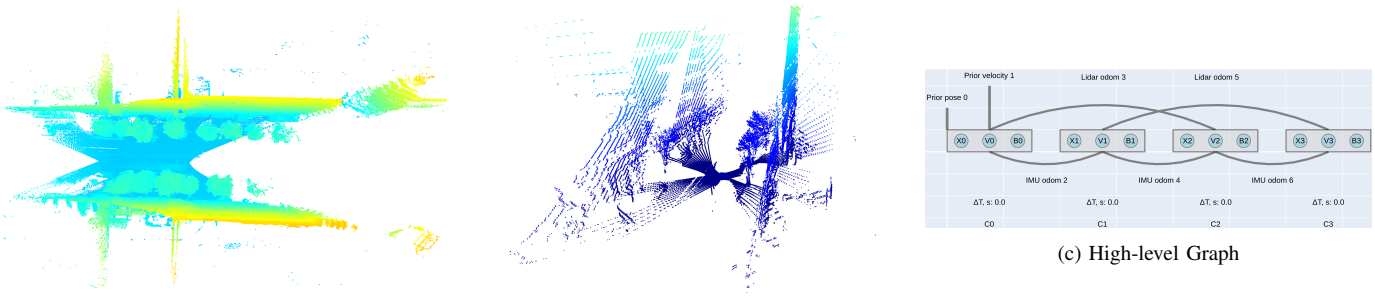


Fig. 5: A High-level graph with one edge and multiple vertices contributing to the corresponding backend factor. The properties of the edge and vertices are enclosed in brackets.

For convenient storage and manipulation of the factor graph [1], we propose a High-level Graph structure. This serves as an interface between the optimizable factor graph and other modules, incorporating additional properties required for versatile localization and mapping tasks. As illustrated in Fig. 5, the High-level Graph consists of vertices and edges. An edge connects $1, ..., N$ vertices, contributing to the calculation of the Backend factor and variables estimation. The base `Vertex` contains a unique index, an estimated value, and a Backend instance with extra properties necessary for solving the optimization problem. This design enables flexible integration with different optimization libraries. However, a vertex can also be non-optimizable, such as a 2D visual feature used for camera tracking. As mentioned in the previous paragraph, the rectangle with three dots signifies that additional properties can be added to the inherited class from the base vertex. The base `Edge` includes a `Backend` factor instance for the estimation process, a set of attached vertices, and a unique index. Additionally, it stores a `Measurement` instance containing the processed measurement and covariance, `Data Batch` elements (without raw data) used to create the measurement, and time properties such as a timestamp and an optional time range for measurements derived from multiple elements. These structures are memory-efficient while providing a convenient way to store and access essential information.



(a) Full map (b) Sparse map (c) High-level Graph

Fig. 6: Bird's-eye view of two point cloud maps created using scans from the left and right LiDARs, an IMU, and a VRS GPS. Map (a) contains 152 scans, while map (b) demonstrates a sparse point cloud created with 4 scans. Image (c) depicts the corresponding high-level graph for the sparse map (b). The scene illustrates a downtown street with trees and buildings surrounding the car.

To group multiple vertices belonging to the same time instance, we introduce a `Vertex Cluster`. If vertices are created for the factors which utilize measurements with the same timestamps, the `Vertex Cluster` will have a timestamp property. However, the vertices created from measurements with different timestamps can be treated as a single entity. In this case a time range attribute which stores the most recent and oldest timestamps of the corresponding measurements is added. Fig. 6 (c) illustrates a High-level Graph with clusters (C_0, C_1, C_2, C_3) containing vertices of different types: `Pose` (P), `Linear Velocity` (V), and `IMU biases` (B).

Under the assumption of a zero-mean Gaussian noise model for N sensor measurements, the MAP inference problem defined in Equation 1 can be factorized due to the proportionality $\phi_i(X_i) \propto \exp\left(-\frac{1}{2}\left\|h_i(X_i) - z_i^k\right\|^2_{\Sigma_i}\right)$ and reduced to the minimization problem:

$$\begin{aligned} X^{\text{MAP}} &= \arg\max_X p(X|Z) \doteq \arg\max_X \phi(X) = \\ &\arg\max_X \prod_{i=1}^N \phi_i(X_i) \arg\min_X -\log \prod_{i=1}^N \phi_i(X_i) \\ &= \arg\min_X \sum_{i=1}^N \|h_i(X_i) - z_i^k\|^2_{\Sigma_i} \end{aligned} \tag{2}$$

where $h_i(X_i)$ is a measurement model, z_i^k is a real measurement, and Σ_i is the measurement noise covariance matrix.

To solve the above problem defined by the factor graph, we utilize GTSAM [13] — a flexible and efficient C++ framework that provides a high-quality Python API. It offers various approaches to obtain a solution such as iSAM2, Levenberg-Marquardt, Dogleg, and others. Additionally, GTSAM benefits from numerous predefined factors, usage examples, and comprehensive documentation. The library includes tools for factor graph serialization and debugging in "dot" and ".text" formats. Additionally, we introduce a custom module for interactive High-level Graph visualization, as illustrated in Fig. 1 and 6.

IV. Experiments

To evaluate the proposed framework and demonstrate its capabilities, we conducted experiments on the Kaist Urban Dataset [18], which features diverse city environments such as highways, streets, and buildings with varying car dynamics. The sensor setup includes a Virtual Reference Station GPS with a solution status of 4 (fixed), left and right LiDARs, and an IMU. For LiDAR-based odometry, we utilize KISS-ICP [19]—a voxel-based scan matching algorithm that computes the transformation $T^{i,j}_{\text{obs}} \in SE(3)$ between two point clouds. Consecutive left-to-left and right-to-right LiDAR scans, without any environment-specific parameter fine-tuning, are used to estimate the odometry between them. Due to the high-quality API of the KISS-ICP library and the proposed design patterns, the odometry computation is implemented in a single class with only a few methods. The IMU measurements are used to create pre-integrated IMU factors [20], and the optimization problem is solved using the Levenberg-Marquardt algorithm in the Backend module. The state vector $X = [x_1, ..., x_N, v_1, ...v_N, b_1, ..., b_N]$ consists of poses $X \in SE(3)$, linear velocities $V \in R^3$, and both accelerometer & gyroscope biases $B \in R^6$. The minimization function derived from Equation 2 is defined as:

$$\begin{aligned} X^{\text{MAP}} = \arg\min_X \Bigg(&\sum_{i=1}^N \|\text{Log}(T^{i,j^{-1}}_{\text{pred}} T^{i,j}_{\text{obs}})\|^2_{\Sigma_i^{\text{Odom}}} \\ &+ \sum_{i=1}^M \|\text{Log}(T^{-1}_{\text{pred}_i} T_{\text{obs}_i})\|^2_{\Sigma_i^{\text{GPS}}} + \sum_{i,j \in \mathcal{K}_k} \|r^{\mathcal{I}}_{i,j}\|^2_{\Sigma_i^{\text{IMU}}} \Bigg) \end{aligned} \tag{3}$$

where $\text{Log}(\cdot)$ is the logarithmic map to convert a transformation $T \in SE(3)$ to a vector in R^6 in the tangent space (Lie algebra $\mathfrak{se}(3)$). $r^{\mathcal{I}}_{i,j}$ — a preintegrated IMU factor between two clusters containing poses, linear velocities, and IMU biases.

Fig. 6 (a) shows a bird's-eye view of the point cloud map created using 152 scans from left and right LiDARs, along with IMU and GPS measurements. Image (b) highlights the sparsity in the scans, which significantly degrades the quality of scan matching. Both LiDARs are mounted at a 45° angle to the car's roof. Image (c) illustrates the corresponding high-level graph for the map in (b). Poses, linear velocities, and IMU accelerometer and gyroscope biases are estimated at the time of the LiDARs' scans. The absence of a fixed GPS solution status and inaccurate LiDAR odometry estimations, especially on highways and in open areas, prevents the creation of a complete map for the dataset without specialized approaches to address these issues. In this paper, we focus more on the architectural aspects of the library rather than

on state estimation quality, which strongly depends on the algorithms used. Nevertheless, the results demonstrate the framework's capability to effectively fuse information from different sensors.

V. Conclusion

In this paper, we have presented a modular and expandable framework for versatile localization and mapping tasks. ModuSLAM enables the fusion of information from different sensors and supports the use of diverse processing algorithms, which can be implemented and integrated as separate modules or services. The proposed software architecture combines both flat service-based and hierarchical tree-based approaches, simplifying the implementation and integration of algorithms for various SLAM challenges. A new data structure, `Data Batch`, introduces the `Element` class, which stores unique location identifiers for raw measurements. This design avoids the need to keep unprocessed data in memory while enabling efficient on-demand retrieval. The High-level Graph enhances the interaction between the modules and estimated variables, making the mapping and localization processes more flexible. The `Vertex Cluster` have been introduced to aggregate variables of different types that belong to the same temporal entities, as defined by the user. The proposed software design patterns have been tested in various mapping scenarios, including LiDAR, LiDAR-inertial, and LiDAR-inertial odometry with GPS anchoring, using the KAIST Urban Dataset. In future work, we plan to add support for more datasets and processing algorithms, targeting applications such as hierarchical and photometric mapping.

References

[1] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, pp. 1–139, 01 2017.

[2] D. Sharafutdinov, M. Griguletskii, P. Kopanev, M. Kurenkov, G. Ferrer, A. Burkov, A. Gonnochenko, and D. Tsetserukou, "Comparison of modern open-source visual SLAM approaches," *Journal of Intelligent & Robotic Systems*, vol. 107, no. 3, p. 43, mar 2023. [Online]. Available: https://doi.org/10.1007/s10846-023-01812-7

[3] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[4] L. Freda, "pyslam: An open-source, modular, and extensible framework for slam," *arXiv preprint arXiv:2502.11955*, 2025.

[5] M. Colosi, I. Aloise, T. Guadagnino, D. Schlegel, B. D. Corte, K. O. Arras, and G. Grisetti, "Plug-and-play slam: A unified slam architecture for modularity and ease of use," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5051–5057.

[6] J. Solà, J. Vallvé, J. Casals, J. Deray, M. Fourmy, D. Atchuthan, A. Corominas-Murtra, and J. Andrade-Cetto, "Wolf: A modular estimation framework for robotics based on factor graphs," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4710–4717, 2022.

[7] A. Cramariuc, L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena, "maplab 2.0 – a modular and multi-modal mapping framework," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 520–527, 2023.

[8] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 298–304.

[9] S. Leutenegger, S. Lynen, M. Bosse, R. Y. Siegwart, and P. T. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, pp. 314 – 334, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:206500609

[10] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.

[11] J. L. Blanco-Claraco, "A flexible framework for accurate lidar odometry, map manipulation, and localization," 2024. [Online]. Available: https://arxiv.org/abs/2407.20465

[12] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, pp. 416 – 446, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:83459364

[13] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022. [Online]. Available: https://github.com/borglab/gtsam

[14] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 10 2023. [Online]. Available: https://github.com/ceres-solver/ceres-solver

[15] G. Grisetti, C. Stachniss, and W. Burgard, "Nonlinear constraint network optimization for efficient map learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 428–439, 2009.

[16] H. Huang, L. Li, H. Cheng, and S.-K. Yeung, "Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular, stereo, and rgb-d cameras," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 21 584–21 593.

[17] T. Qin and S. Shen, "Online temporal calibration for monocular visual-inertial systems," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3662–3669.

[18] J. Jeong, Y. Cho, Y.-S. Shin, H. Roh, and A. Kim, "Complex urban dataset with multi-level sensors from highly diverse urban environments," *International Journal of Robotics Research*, vol. 38, no. 6, pp. 642–657, 2019.

[19] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, pp. 1029–1036, 2023.

[20] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *Trans. Rob.*, vol. 33, no. 1, p. 1–21, Feb. 2017. [Online]. Available: https://doi.org/10.1109/TRO.2016.2597321